STANFORD UNIVERSITY, STANFORD, CALIFORNIA 94305

ROBERT H. CANNON, JR.
CHARLES LEE POWELL PROFESSOR
DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS

DURAND BUILDING, ROOM 275
TELEPHONE: (415) 723-3601
FAX: (415) 725-3377
E-MAIL: CANNON@SIERRA.STANFORD.EDU

~~June 14, 1995~~

Scientific Officer Code: 1133
Dr. Ralph Wachter
Office of Naval Research
800 North Quincy Street
Arlington, VA 22203-1714

Re: Department of the Navy Ref. N00014-92-J-1809

Dear Dr. Wachter:

Enclosed herewith are quarterly progress reports covering research on "ControlShell: A Real-Time Software Framework," for the periods October-December 1993, ~~January-March 1994, April-June 1994, July-September 1994, and October-December 1994.~~

We thank you for your interest in and valuable support of this research.

Sincerely yours,

Robert H. Cannon

Encl

cc: Dr. Pradeep Khosla
Administrative Grants Officer
Director, Naval Research Laboratory
Defense Technical Information Center ✓
Ruth Kaempf, SPO, Stanford

DEPARTMENT OF THE NAVY
OFFICE OF NAVAL RESEARCH
SEATTLE REGIONAL OFFICE
1107 NE 45TH STREET, SUITE 350
SEATTLE WA 98105-4631

IN REPLY REFER TO:
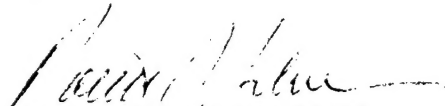
4330
ONR 247
11 Jul 97

From: Director, Office of Naval Research, Seattle Regional Office, 1107 NE 45th St., Suite 350, Seattle, WA 98105

To: Defense Technical Center, Attn: P. Mawby, 8725 John J. Kingman Rd., Suite 0944, Ft. Belvoir, VA 22060-6218

Subj: RETURNED GRANTEE/CONTRACTOR TECHNICAL REPORTS

1. This confirms our conversations of 27 Feb 97 and 11 Jul 97. Enclosed are a number of technical reports which were returned to our agency for lack of clear distribution availability statement. This confirms that all reports are unclassified and are "APPROVED FOR PUBLIC RELEASE" with no restrictions.

2. Please contact me if you require additional information. My e-mail is *silverr@onr.navy.mil* and my phone is (206) 625-3196.

ROBERT J. SILVERMAN

Darpa/Navy Contract No. N00014-92-J-1809

ControlShell: A Real-Time Software Framework

Quarterly Progress Report — October – December, 1993

P.I.'s: Prof. J.C. Latombe, Prof. R.H. Cannon, Jr, Dr. S.A. Schneider

Executive Summary

We are creating a new paradigm for building and maintaining complex real-time software systems for the control of moving mechanical systems. This objective is being met through the *simultaneous* development of both a powerful software environment and cogent motion planning and control capabilities. Our research concentrates on three key areas:

- Building an innovative, powerful real-time software framework,
- Implementing new distributed control architectures for intelligent mechanical systems, and
- Developing distribution architectures and new algorithms for the computationally "hard" motion planning and direction problem.

Perhaps more importantly, we are working on the *vertical integration* of these technologies into a powerful, working system. It is only through this coordinated, cooperative approach that a truly revolutionary, usable architecture can result.

Summary of Progress

This section highlights some of our achievements for this quarter. During this period, we have:

- Added distributed-query facility to NDDS.
- Performed initial design and class-hierachy specification for the new (C++ based) version of ControlShell.
- Developed and implemented control algorithms to enable stable manipulator control through kinematic singularities, and cooperative object control.
- Developed sensor integration approach to achieve robust tracking of objects on the conveyor belt.
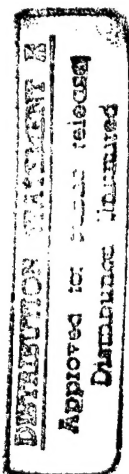
Chapter 1

Introduction

The goal of this research project is to build a new paradigm for building and maintaining complex real-time software systems for the control of moving mechanical systems. This objective is being met through the *simultaneous* development of both a powerful software environment and cogent motion planning and control capabilities. Our research concentrates on three areas:

- Building an innovative, powerful real-time software development environment,

- Implementing a new distributed control architecture, and using it to deftly control and coordinate real mechanical systems, and

- Developing a computation distribution architecture, and using it to build on-line motion planning and direction capabilities.

We believe that no technology can be successful unless proven experimentally. We are thus validating our research by direct application in several disparate, real-world settings.

This concurrent development of system framework, sophisticated motion planning and control software, and real applications insures a high-quality architectural design. It will also embed, in *reusable* components, fundamental new contributions to the science of intelligent motion planning and control systems. Researchers from our three organizations, the Stanford Aerospace Robotics Laboratory (ARL), the Stanford Computer Science Robotics Laboratory (CSRL), and Real-Time Innovations, Inc. (RTI) have teamed to cooperate intimately and directly to achieve this goal. The potential for advanced technology transfer represented by this cooperative, vertically-integrated approach is unprecedented.

Framework Development This research builds on an object-oriented tool set for real-time software system programming known as *ControlShell*. It provides a series of execution and data interchange mechanisms that form a framework for building real-time applications. These mechanisms

- Defined and implemented a new path planning method (the vector-based planner) to generate paths for robots with many degrees of freedom.

- Partially designed and implemented a new, two-phase path-planning approach, which we call the "randomized roadmap planner."

- Designed and implemented a randomized three-arm manipulation planner for manipulating an elongated object in a 3D cluttered environment.

- Completed development of a powerful multi-mobile-robot simulator to facilitate the development and debugging of programs for multiple interacting mobile robots.

Our research is progressing according to schedule.

are specifically designed to allow a component-based approach to real-time software generation and management. By defining a set of interface specifications for inter-module interaction, ControlShell provides a common platform that is the basis for real-time code exchange and reuse.

Our research is adding fundamental new capabilities, including network-extensible data flow control and a graphical CASE environment.

Distributed Control Architecture   This research combines the high-level motion planning component developed by the previous effort with a deft control system for a complex multi-armed robot. The emphasis of this effort is on building interfaces between modules that permit a complex real-time system to run as an interconnected set of distributed modules. To drive this work, we are building a dual-arm cooperative robot system that will be able to respond to high-level user input, create sophisticated motion and task-level plans, and execute them in real time. The system will be able to effect simple assemblies while reacting to changing environmental conditions. It combines a world modelling system, real-time vision, task and path planners, an intuitive graphical user interface, an on-line simulator, and sophisticated control algorithms.

Computation Distribution Architecture   This research thrust addresses the issues arising when computationally complex algorithms are embedded in a real-time framework. To illustrate these issues we are considering two particular problem domains: *object manipulation by autonomous multi-arm robots* and *navigation of multiple autonomous mobile robots in an incompletely known environment*. These two problems raise a number of generic issues directly related to the general theme of our research: motion planning is provably a computationally hard problem and its outcomes, motion plans, are executed in a dynamic world where various sorts of contingencies may exist.

The ultimate goals of our investigation are to both provide real-time controllers with on-line motion reactive planning capabilities and to build experimental robotic systems demonstrating such capabilities. Moreover, in accomplishing this goal, we expect to identify general guidelines for embedding a capability requiring provably complex computations into a real-time framework.

Chapter 2

ControlShell Framework Development

This section describes our progress in developing the ControlShell framework and underlying architecture. Two fundamental extensions to ControlShell are being pursued:

- Distributed information sharing paradigms, by Gerardo Pardo-Castellote and Stan Schneider.

- Graphical Computer Aided Software Engineering (CASE) environments, by Stan Schneider and Vince Chen.

2.1 Distributed Information Sharing Paradigms: NDDS

Through customer interaction, we have become aware of the need to provide a "distributed-query" facility. certain data doesn't change frequently (ever) or is not consistently required so that despite the larger latency, a query is a more appropriate model for the data flow.

It is not trivial to provide clear semantics for a distributed query facility there there may be multiple producers of the same data instance. You issue a query and d several people respond. Which one should you choose, how long should you wait for additional data when do you give up if nobody responds etc.

Our approach has been to provide the user with a flexible mechanism that model the physical situation so that it may be tailored to the application's needs. Multiple producer conflict resolution is achieved through the use of producer strengths and persistence (just as in the subscription case). There user can balance the tradeoff between getting data fast and getting the "best" available data with the help of two parameters: waitTime and deadline. The user will bet the response received by the strongest producer within the wait-time, or the first response after the waitTime before

the dead-line. If no response is received by the dead-line the query returns with an error. Setting this parameters the user may choose to wait indefinitely(deadline is infinite), get the first response (waitTime=0) or any intermediate situation. These times can be altered by the user at any time.

To support the distributed query, NDDS maintains a replicated database of who is producing what. This database has the same characteristics as the "who wants what" database and in fact uses the same mechanisms for its maintenance and distribution. In particular, it is fully symmetrically distributed, and aged so that productions no longer active are eventually removed.

2.2   ControlShell CASE Environment

2.2.1   Object-Oriented Design

This quarter, we began the redesign of ControlShell to take advantage of the object-oriented features of C++. ControlShell's already modular and object-oriented design makes this transition relatively easy.

2.2.2   Module Classes

CSModules   The ControlShell run-time architecture is based on a simple concept: the *CSModule*. A CSModule is a named routine with pointers to data already bound to it. The system can find and execute any CSModule at any time without needing to supply the CSModule with its data.

The `CSModuleClass` is the abstract base class of all ControlShell execution modules. It binds a name to an execution routine (pure virtual function). Instance classes are expected to define the actual execution code as well as any data it needs for execution.

CSSampleModules   A *CSSampleModule* extends upon the CSModule by binding multiple routines, each of which is executed at well-defined times. Additionally, each CSSampleModule contains lists of input and output signal dependencies allowing CSSampleModules to be sorted to determine the order of execution.

`CSSampleModuleClass` is also an abstract base class derived from `CSModuleClass`. It binds other execution routines that can be expected of a module that executes on a sample list. These additional methods include `stateUpdate`, `enable`, `disable`, `startup`, `shutdown`, `timingChanged`, `terminate`, and `reset`. Instance classes only need to define and implement the methods it needs. If not defined, they default to null routines.

Component Classes   *CSComponents* are yet again derived from CSSampleModules, providing methods to print the data structure bound to the component—in formats for human or machine

consumption. The `CSComponentClass`, derived from `CSSampleModuleClass`, serves as the base class for all ControlShell components.

Each component is implemented as a separate derived class. The derived class definition is automatically generated from the Component Editor's description of the component's data flow requirements. Thus, the data structure and skeleton code for the required methods for the component is automatically built for the user. The generated code also includes methods to parse data-flow description files (generated by the DFE editor) and to instance new objects that implement components.

2.2.2.1 Execution List Classes

CSSampleLists ControlShell uses *CSSampleLists* to manage the execution of CSSampleModules. A CSSampleList contains a list of registered CSSampleModules that are to be sequentially executed. Based on the trigger that starts the execution sequence, the CSSampleList determines which of the CSSampleModule's routines to run. For example, at "Sample" time, every (enabled) module's `execute` routine is executed, then every module's `stateUpdate` routine is executed. This design was detailed in last quarter's report, and in the attached DFE user's manual.

`CSSampleListClass` is the base class that provides facilities for registering CSSampleModules and methods that can be called at these "well-defined" times to execute the proper module routines. The `CSSampleListClass` and its subclasses are internal to ControlShell and are not meant to be directly manipulated by the user.

CSSampleHabitats Finally, a *CSSampleHabitat* is derived from CSSampleList to provide a named sampled-data environment. A CSSampleHabitat encapsulates all the information and defines all the interfaces required for sampled-data programs to co-exist. It also contains routines to control the sampling process and timing source. For example, a module installed into a sample habitat can query its clock source and sample rate, start and stop the sampling process, etc.

Each sample habitat contains an independent task that executes the sample code. The task is clocked by the periodic source (such as a timer interrupt). Additionally, the execution order in a `CSSampleHabitatClass` is automatically determined by sorting the CSSampleModules (and their derived Component classes) according to their input and output dependencies.

The ControlShell structure described here becomes quite amenable for implementation using C++. The ControlShell class structure consists of a fairly shallow tree to allow users to develop ControlShell components quickly and painlessly, without having to dig through the inheritance tree. Moreover, the automatic code generation of the ControlShell Component Editor further shortens development time.

2.2.3 Configuration Management

During this quarter, we also made good progress in designing the configuration management capabilities of ControlShell.

Complex real-time systems often have to operate under many different conditions. The changing sets of conditions may require drastic changes in execution patterns. For example, a robotic system coming into contact with a hard surface may have to switch in a force control algorithm, along with its attendant sensor set, estimators, trajectory control routines, etc.

ControlShell's configuration manager directly supports this type of radical behavior change; it allows entire groups of modules to be quickly exchanged. Thus, different system personalities can be easily interchanged during execution. This is a great boon during development, when an application programmer may wish, for example, to quickly compare controllers (See Figure 2.1). It is also of great utility in producing a multi-mode system design. By activating these changes from the state-machine facility (see previous reports), the system is able to handle easily external events that cause major changes in system behavior.

Configuration Hierarchy The configuration manager essentially creates a four-level hierarchy of module groupings. Individual sample modules form the lowest level. These usually implement a single well-defined function. Sets of modules, called *module groups*, combine the simple functions implemented by single modules into complete executable subsystems.

Each module group is assigned to a *category*. One group in each installed category is said to be *active*, meaning its modules will be executed. Finally, a *configuration* is simply a specification of which group is active in each category.

Example As a simple example, consider a system with two controllers: a proportional-plus-derivative controller named "PD", and an optimal controller known as "LQG". Suppose the PD controller requires filtered inputs, and thus consists of two sample modules: an instance of the *PDControl* component and a filter component. These two components would comprise the "PD" module group. The "LQG" controller module group may also be made up of several components. Both of these groups would be assigned to the category "controllers".

The user (or application code) can then easily switch controllers by changing the active module group in the "controller" category.

Now suppose further that the controllers require a more sophisticated sensor set. A category named "sensors" may also be defined, perhaps with module groups named "endpoint" and "joint". The highest level of the hierarchy allows the user to select an active group from each category, and name these selections as a *configuration*. Thus, the "JointPD" configuration might consist of the "joint" sensors and the "PD" controller. The "endptLQG" configuration could be the "endpoint" sensors and the "LQG" controller.
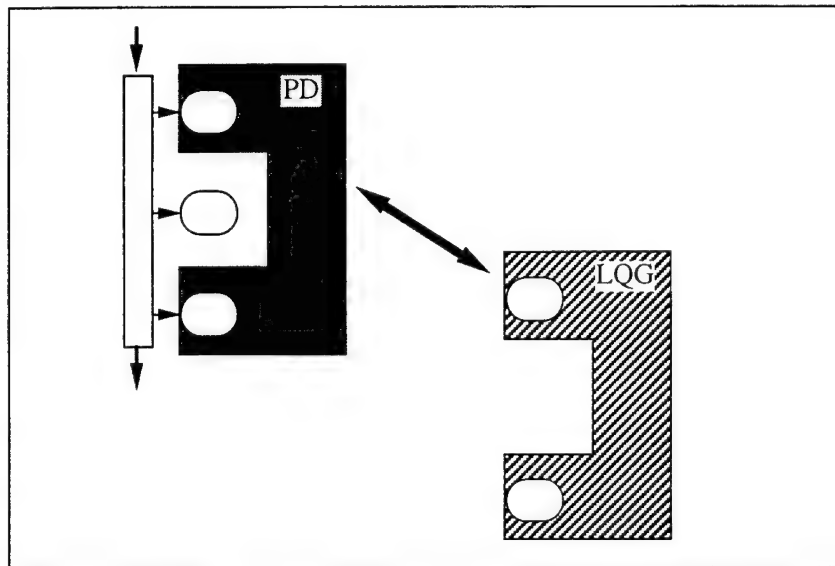
Figure 2.1: **Configuration Manager**

Configurations can be swapped in or out under program or menu control. This provides flexible run-time reconfiguration of the execution structure.

Category and Group Specification This subdivision may seem complex in these simple cases. However, it is quite powerful in more realistic systems. It has been shown to be quite natural in our experimental applications. Our work this quarter has focused on implementing this structure within the new graphical editing tools.

Assigning modules to groups and groups to categories is made quite simple with the ControlShell graphical DFE editor's "configuration definition" window, shown in Figure 2.2. New categories are added with the click of a button. To create a module group, the user simply names a group, and then clicks on the modules in the data-flow diagram that should belong to that group. The blocks are color-coded to relate the selections back to the user.

2.2.3.1 Configuration Class Design

ControlShell uses three classes, CSModuleGroups, CSCategory's, and CSConfigurations to manage groups of CSSampleModules.

A CSModuleGroup contains a group of closely-coupled modules that must work together, and a CSCategory defines a set of CSModuleGroups, only one of which can be active at any one time. A CSConfiguration is defined as the set of active CSModuleGroups—one from each CSCategory.

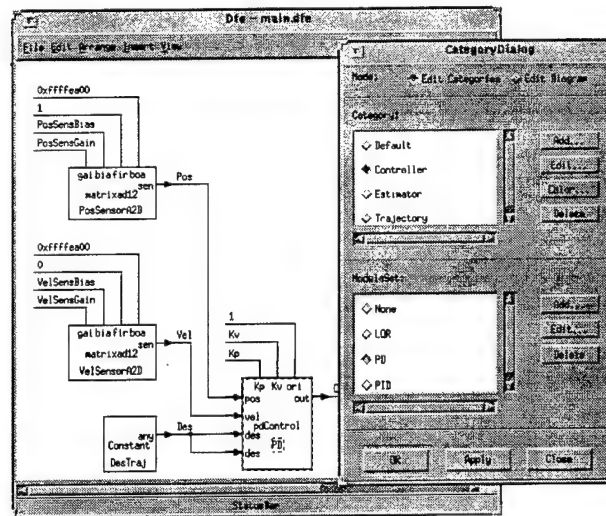The top-level construct, CSConfiguration, defines a complete set of modules in the system that

Figure 2.2: **Configuration Definition**

Configurations are easily defined within the DFE graphical interface.

is to be enabled to implement an operational mode. A configuration consists of the selection of one active module group from each category. Changing CSConfigurations changes the operational mode of the entire system.

In the example above, enabling a CSConfiguration may bring in a new controller, sensor set, estimator, trajectory control routines, safety modules, *etc.*—everything required for the new mode of operation. For instance, suppose a user has defined CSCategories named "Controller", "Estimator", "Filter", and "Sensor". CSModuleGroups with the "Controller" CSCategory may include "PDControl", "PIDControl", and "LQRegulator", each of which contains the CSSampleModules needed to perform that function. The "LQ" CSConfiguration, therefore, might be defined as:

- LQRegulator module group from the "Controller" category

- LQEstimator module group from the "Estimator" category

- JointSensors module group from the "Sensors" category

- And no (empty) module group from the "Filter" category

We should note here that CSModuleGroups may contain overlapping individual modules. This allows arbitrary configurations of modules to be active in any CSConfiguration, a powerful and

useful concept. For instance, the JointSensors module group above may contain the actual Analog-to-Digital device driver module; that module could also be a part of every other module group in the "Sensors" category.

Chapter 3

Distributed Control Architectures and Interfaces

This section covers our research in software architectures, communication protocols and interfaces that will advance the state-of-the-art in the prototyping-development-testing cycle of high-performance distributed control systems. These interfaces will be implemented within the framework described in Chapter 2. The results of this research will be applied to the vertical integration of planning and control and demonstrated by executing a set of challenging tasks on our two-armed robot system.

There are three main thrusts to this research:

- Development of inter-module interfaces for distributed control systems, by Gerardo Pardo-Castellote.

- Development of a control methodology capable of executing high-level commands, by Gerardo Pardo-Castellote, Tsai-Yen Li, and Yotto Koga.

- Hardware development and experimental verification, by Gerardo Pardo-Castellote and Gad Shelef.

This quarter, we devoted considerable attention to the manipulator control issues, specifically control though a kinematic singularity and cooperative object control.

3.1   Inter-Module Interfaces

The inter-module interfaces have remained unchanged during this quarter.

3.2   Control Methodology Development

During this quarter we have achieved the following tasks:

- Arm control through a singularity.

- Cooperative object control.

- Robust tracking of objects on the conveyor.

3.2.1   Arm control through a singularity

The control achieved at the joint-level allows the arm-control above it to treat the manipulator as if it had ideal motors and the commanded torques were delivered perfectly to the rigid links.

Ultimately any arm controller must generate joint torque commands based on the manipulator's desired state; however, different controllers differ in two related but somewhat decoupled issues:

- The error law or control policy: selection of the space in which errors are computed and specification of reference behavior when reacting to external disturbances. Typical choices include position and velocity errors in Cartesian or joint space, impedance relationships and hybrid position/force control schemes.

- The implementation of the policy. The type of controller used to enforce the error law. Typical choices are inverse dynamics (also referred as computed-torque) methods, kinematic controllers and independent-joint-space controllers.

Inverse dynamics controllers, which are the highest performance control schemes, require a good kinematic and dynamic model of the manipulator. The following sections describe the control scheme used for the workcell manipulators.

The arm controller used in the Manufacturing Workcell combines two controllers: a high-performance inverse-dynamics controller used for manipulator-configurations away from kinematic singularities and a low-performance (yet robust) joint-space PID controller used near singularities. These two controllers are blended in a transition region.

The high-performance controller implements an impedance relationship at an operational point located at the center of the tool (gripper), and uses an inverse dynamics (computed-torque) approach to enforce this relationship. This particular choice of control law (policy) and implementation is usually referred as "arm impedance control." The advantages of arm-impedance control in the context of flexible drive-train robots and object manipulation have been described in previous research by Schneider and Pfeffer.

Symbol	Dimension	Meaning
$\mathbf{M_{imp}}$	4×4	Desired virtual mass (diagonal matrix).
$\mathbf{x}, \dot{\mathbf{x}}, \ddot{\mathbf{x}}$	4×1	Actual Cartesian position, velocity and acceleration of the operational point.
$\mathbf{x_{ref}}, \dot{\mathbf{x}}_{\mathbf{ref}}, \ddot{\mathbf{x}}_{\mathbf{ref}}$	4×1	Reference Cartesian position, velocity and acceleration of the operational point.
$\mathbf{K_p}, \mathbf{K_v}$	4×4	Desired virtual stiffness and damping (diagonal).
$\mathbf{f_{ext}}$	4×1	External force acting on the operational point.
$\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$	4×1	Vector of generalized coordinates and time derivatives.
$\mathbf{M(q)}$	4×4	Configuration-dependent mass matrix.
$\mathbf{B(q)}$	4×3	Configuration-dependent matrix of Coriolis term.
$\mathbf{C(q)}$	4×4	Configuration-dependent matrix of centripetal terms.
$\mathbf{g(q)}$	4×1	Configuration-dependent gravity vector.
$\mathbf{J(q)}$	4×1	Jacobian relating joint-rates to operational-point velocities.
τ	4×1	Torque vector.

Table 3.1: **Notation for arm-impedance controller derivation.**

The above sizes apply to a 4-DOF SCARA manipulator. The cartesian vector **x** *contains the position and rotation about the vertical axis of the operational point.*

The structure of the arm-impedance controller is described below for the simpler case of a SCARA manipulator. Table 3.1 summarizes the meaning of the different symbols used in the description.

First, the error law specifies that the manipulator operational point (normally the end-effector or tool) must obey the following impedance relationship:

$$M_{imp}\ddot{x} = M_{imp}\ddot{x}_{ref} + K_p(x_{ref} - x) + K_v(\dot{x}_{ref} - \dot{x}) + f_{ext} \tag{3.1}$$

Second, given that the actual equation of motions of the manipulator:

$$M(q)\ddot{q} + B(q)[\dot{q},\dot{q}] + C(q)[\dot{q}^2] + g(q) + J^t(q)f_{ext} = \tau \tag{3.2}$$

And that the cartesian coordinates of the operational-point are related to the configuration-space coordinates through the manipulator Jacobian:

$$\dot{x} = J(q)\dot{q} \quad \Rightarrow \quad \ddot{x} = J(q)\ddot{q} + \dot{J}(q)\dot{q} \Rightarrow \quad \ddot{q} = J(q)^{-1}(\ddot{x} - \dot{J}(q)\dot{q}) \tag{3.3}$$
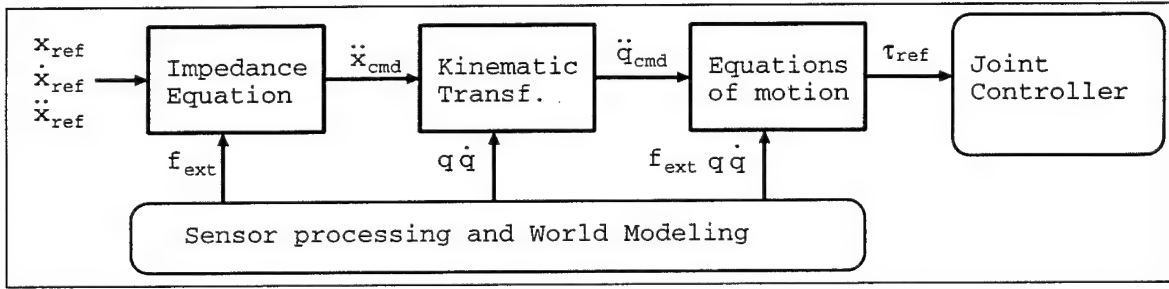

Figure 3.1: **Block diagram for the computed torque impedance controller.**

The arm controller enforces an impedance relationship at the arm operational point (OP). Given the cartesian error between the reference (desired) and actual states for the OP, the impedance relationship specifies the commanded acceleration $\ddot{x}_{com}$ for the OP (i.e. the acceleration that will satisfy the impedance relationship). This acceleration is transformed into the equivalent joint-accelerations $\ddot{q}_{cmd}$ using the manipulator kinematics. Given $\ddot{q}_{cmd}$, the manipulator dynamics are used to determine the reference joint torques τ_{ref} that will achieve this joint-acceleration. These joint torques become the reference command for the joint-control layer below.

The impedance relationship (3.1) can be enforced if actuator torques are chosen in (3.2) such that the resulting acceleration $\ddot{q}$ (equation (3.3)) results in operational-point accelerations $\ddot{x}$ which verify the impedance relationship. This scheme is illustrated in Figure 3.1.

The disadvantage of this inverse-dynamics controller is that it cannot be used to transition the arm through a kinematic singularity (which occurs whenever the elbow is fully extended). At the
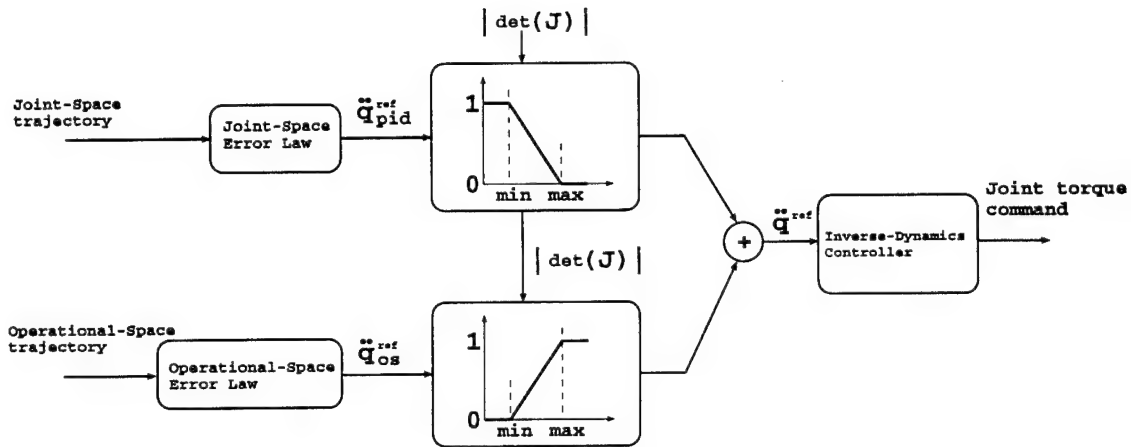
Figure 3.2: **Merging a cartesian-space computed-torque controller with a joint-space PID controller.**

The strategic module is required to specify both the cartesian-space and the corresponding joint-space trajectories for any arm motion. These trajectories are run in parallel through a cartesian-space error law (see Figure 3.1 and a joint-space error law. The commanded joint accelerations $\ddot{q}$ from the two controllers are combined to generate the actual joint-acceleration command $\ddot{q}^{cmd}$ for the inverse dynamics controller. The combination is a weighted sum with weights that depend on how close the manipulator is to a singularity as measured by the value of the determinant. The transition interval corresponds to relative elbow angles $|q_{el}|$ between 0.1 and 0.2 radians

kinematic singularity, the Jacobian matrix becomes singular and its inverse cannot be computed as required in equation (3.3). The inverse-dynamics controller was modified so it would be stable even at the singularity by using a Jacobian pseudo-inverse in equation (3.3) (i.e. whenever the determinant of the Jacobian is below a threshold, the pseudo-inverse is used instead of the Jacobian inverse). Although this modified controller works well in regulation, it cannot be used to follow reference trajectories which cross the singularity because these trajectories cannot be specified exclusively in operational-space. In essence, the workcell manipulators are redundant. There are two arm configurations ("elbow in" and "elbow out") which result on the same cartesian location of the operational point. Two different (joint-space) trajectories that start from the same initial state, go through the singularity, and come out with opposite elbow configurations are indistinguishable to the operational space controller.

It is important to be able to cross the singularity under control to take full advantage of the arm workspace. For this reason a hybrid approach was developed: Commanding an arm trajectory requires both the operational-space trajectory and the corresponding joint-space trajectory to be simultaneously specified. The controller monitors the proximity of the arm to a singularity (by evaluating the determinant of the Jacobian matrix). Away from the singularity the inverse-dynamics controller is used. Near the singularity, a pure joint-PID controller is used. In the transition region, a weighted combination of the reference commands from both controllers is used. In either case, the commanded joint-space accelerations are fed to the inverse-dynamics controller as illustrated in Figure 3.2.

A word on the selection of impedances and virtual masses: Note that for $f_{ext} = 0$ only the ratios K_p/M_{imp} and K_v/M_{imp} are relevant. In view of this, these ratios were chosen (through an iterative process) close to the stability limit (for maximum performance), but leaving the system slightly over-damped (overshooting can cause collisions). When the arm is in contact with the environment, the force/torque sensors measure f_{ext} and the operational point is commanded to respond with the same acceleration as a mass of value M_{imp}. For this reason, larger M_{imp} results in more stable, yet "slower" contact behavior. The value finally chosen was a compromise for the required tasks. Gains for joint-space error law were selected such that they produced commanded accelerations of similar magnitude as the operational-space error law for small joint errors at the singularity boundary. In other words, $K_p^{pid} = J(q) K_p/M_{imp}$ and $K_v^{pid} = J(q) K_v/M_{imp}$ at this boundary.

Figure 3.3 illustrates the implementation of the arm-level controller in terms of reusable software components. The full inverse-dynamics computed-torque controller for both arms was run at 200 Hz on a dedicated processor board[1]. Experimental results are presented in section 3.3.

This research does not make the claim that the approach followed here to achieve stable performance through the singularity is valid in general. Merging a singularity-free (yet low performance) controller with a high-performance controller in such a way that the singularity-free controller is used in proximity to the singularity is simple and intuitively appealing (in fact is an approach commonly followed by so called fuzzy controllers). However, there are a multitude of issues to

---

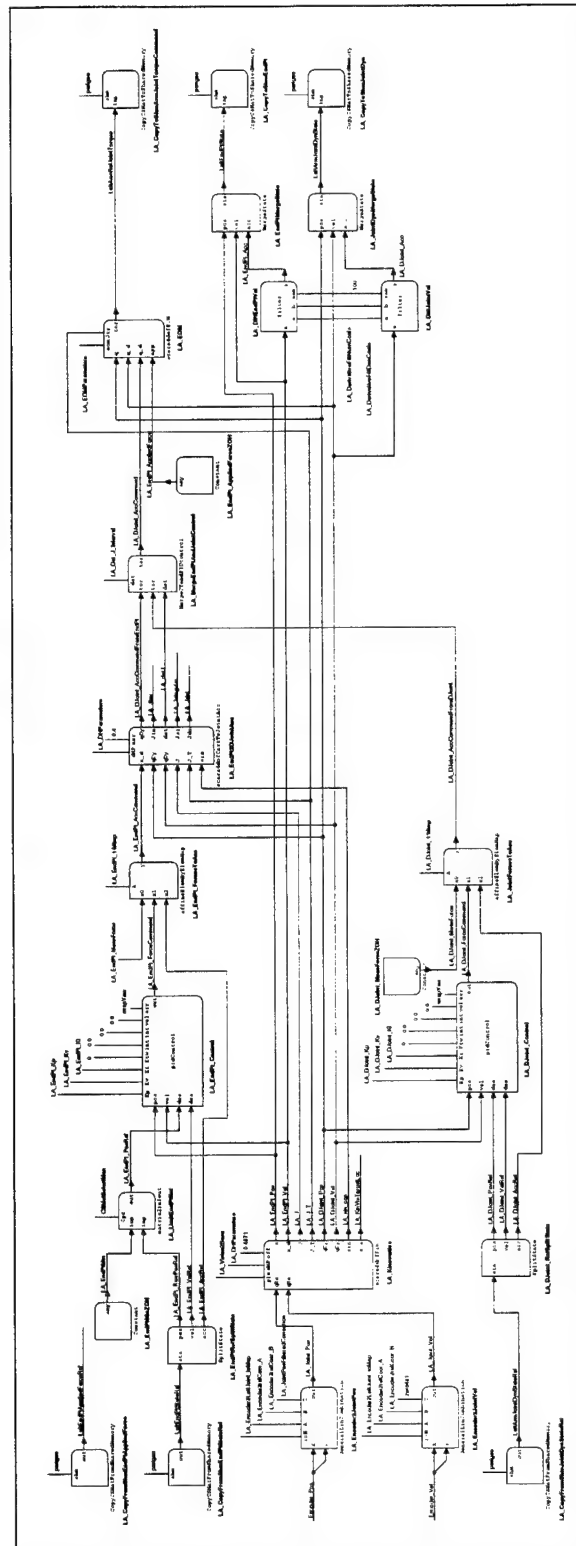[1]A VME-based single-processor computer containing a 33 MHz m68040 processor

Figure 3.3: **Arm-control layer data-flow.**

be addressed more formally for this approach to be rigorous. It is not even clear under which circumstances a weighted (configuration-dependent) average of two controllers results in a stable controller. This investigation could be the topic of continuing research.

3.2.2   Cooperative Object Control

The purpose of the object-control layer is to allow the workcell to manipulate objects either single-handedly or cooperatively using both manipulators. In this section, the term *cooperative* control is used in contrast with *coordinated* control. In either case an object reference trajectory is specified, but in coordinated control, that trajectory is transformed (through the grasp kinematics) into reference arm trajectories, and each individual arm is controlled from its own trajectory (regardless of what the other arm, or the object are doing). In cooperative control no individual-arm trajectories are computed. Rather, the arm commands are computed directly from the object trajectory and error. This closes another control loop on the object state. Figure 3.4 contrasts these approaches. True cooperative object control often requires force sensing at the arm end-effectors so that the interaction forces can be accurately controlled.

There have been many approaches to cooperative object control. This experiment uses the *Object Impedance Control* (OIC) approach originally developed by Schneider. This approach draws from Hogan's impedance control concept and the work of Nakamura. OIC was originally developed for fixed manipulators handling a rigid object. This work was later extended to manipulators on a mobile base by Vasquez, Ullman and Dickson and manipulation of objects with internal dynamics by Meer (all these researchers are at the Stanford Aerospace Robotics Laboratory). These researchers have demonstrated the utility of the OIC approach to cooperative-object manipulation.

The OIC is a model reference controller which enforces an impedance relationship on the state (position, velocity, and acceleration) of a certain point in the object (the object's Remote Center of Compliance or RCC)[2]:

$$M_{\text{imp}}\ddot{x} = M_{\text{imp}}\ddot{x}_{\text{ref}} + K_p\left(x_{\text{ref}} - x\right) + K_v\left(\dot{x}_{\text{ref}} - \dot{x}\right) + f_{\text{ext}} \qquad (3.4)$$

The above equation can be interpreted as representing the equations of motion of a *virtual* object which is affected both by the forces applied to the object f_{ext} and a virtual force which makes it behave as if it were attached to the reference trajectory by a spring and dash-pot on each degree of freedom.

The data-flow of the OIC is illustrated in Figure 3.5. First the RCC and a reference "state" for the RCC, $(x_{\text{ref}}, \dot{x}_{\text{ref}}, \ddot{x}_{\text{ref}})$ are specified. Given the the actual object state $(x, \dot{x})$, and the rigid-body transformation to the RCC, the impedance relationship of equation (3.4) can be used to determine $\ddot{x}$

---

[2]This point does not have to be physically in the body. It is sufficient that it remains fixed in any body-fixed reference frame. That is, it is related through a rigid-body transformation to the object's position.
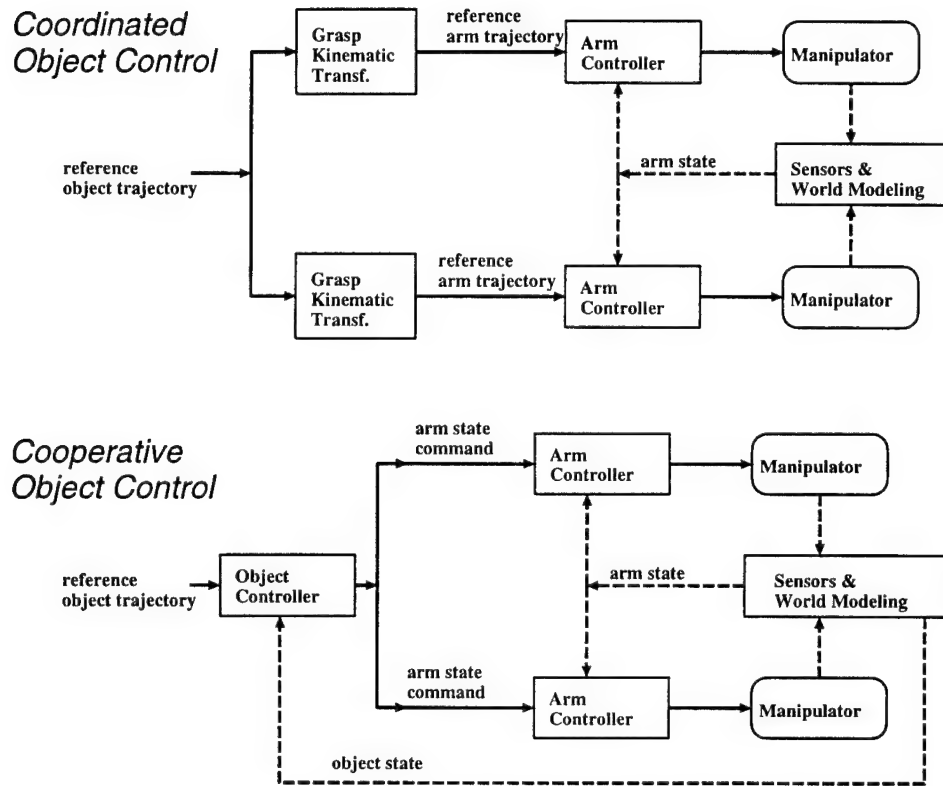
Figure 3.4: **Comparison of cooperative object control with coordinated object control**

In coordinated object control the reference trajectory for the object is transformed using the grasp kinematics into reference trajectories for each arm. Each arm is then controlled independently from its own trajectory. Cooperative object control generates no arm trajectories directly. Instead, arm commands are generated from the reference object trajectory and the error between the trajectory and the actual object state. Cooperative object control closes another control loop on the object state.

Figure 3.5: **Data-flow of the object-impedance controller.**

Implementation of the object-control layer using ControlShell.4.x components. This diagram illustrates the control loop for the "barWM" object. The reference trajectory barWMDesState *can originate from a planned via-point trajectory (*VIP_Generate *component, or a locally generated intercept (*fifthOrderStateTrajectory*) or a tracking (*WM_TrackConveyorObject*) trajectory. The* ObjImpControl *components combine the reference state, actual state, and (estimated) external force on the object to generate reference states and applied forces for each one of the arms. The* ObjImpControl *component, uses the specific grasp transforms, mass properties, and virtual object behavior set by the strategic-layer (using the private world-model interface) to implement the desired impedance relationship.*

(the acceleration that the RCC should have to satisfy the impedance relationship). The acceleration of the RCC $\ddot{\mathbf{x}}$ can now be transformed back to obtain the required object acceleration $\ddot{\mathbf{x}}_{com}$. Since the actual EOM of the body and the grasp transforms are known, the required manipulator end-effector forces and accelerations can be computed. These required end-effector accelerations/torques then become commands for the arm-level controller. In case there is redundancy solving the required manipulator forces on the object, the redundancy can be used to control the internal forces the object is subject to.

The above OIC formulation is restricted to situations where the manipulator arms holding the object are not at a singular configuration. This restriction is enforced by the strategic-control layer.

A nice feature of the OIC approach is that the EOM of the object decouple the arms so that, assuming each arm can compute locally its own required reference end-effector state and applied forces[3], each arm would only need to know about its own dynamics to control itself. This makes the algorithm easily parallelizable and extensible to multiple independent robots which cooperate in the manipulation. The work of Dickson addresses these issues.

3.2.3 Robust Tracking of Objects on the Conveyor

For the workcell to operate properly, it must predict and/or sense the object motions on the conveyor accurately. As previously mentioned, the object is often lost during the capture maneuver because the arm obstructs the camera view of the object's LEDs. This problem is solved by adding LED's to the conveyor (so its location can be determined) and an incremental encoder to the motor driving the conveyor belt (so that belt displacement can be measured)[4]. The World Modeler uses a dedicated sensor-integration software-component per conveyor/object pair. The position and size of the conveyor and various objects is used by the component to determine whether a specific object is on top of that conveyor. If an object is on the conveyor, vision information is used whenever available, otherwise, the object's position and velocity are estimated from the last vision update and the conveyor measurements. This estimation uses the measured conveyor orientation (determined by the vision system) and the relative displacement of the conveyor-belt from the time visual tracking was lost. Object velocity is simply deduced from conveyor speed and orientation. This scheme is illustrated in Figure 3.6. Experimental results are presented in section 3.3

3.3 Hardware Development and Experiments

Figures 3.7 and 3.8, illustrate control system performance when the arm traverses a kinematic singularity. The hybrid controller is able to control the arm motion smoothly through the transition.

---

[3]This computation transforms the required object acceleration through the known grasp transforms and distributes the required force/torque on the object among the two arms.

[4]The procedure described here can be applied to any number of conveyors present in the workcell.
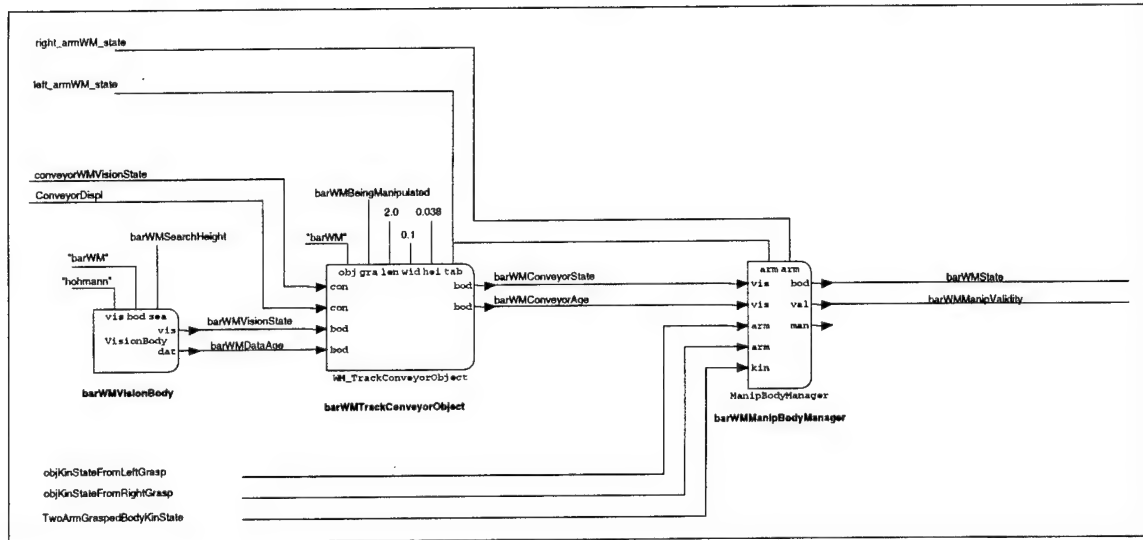
Figure 3.6: **Sensor integration scheme for objects on a conveyor**

A dedicated component exists for each object-conveyor pair. The component uses the vision-sensed position of the conveyor and object to determine whether the object is on the conveyor by modelling the area the conveyor covers and figuring out whether the center of gravity of the object lies on this area. The vision measurements are used whenever available. When vision-tracking is lost, the position of the object is extrapolated from the last visible position using the relative conveyor displacement and its orientation.
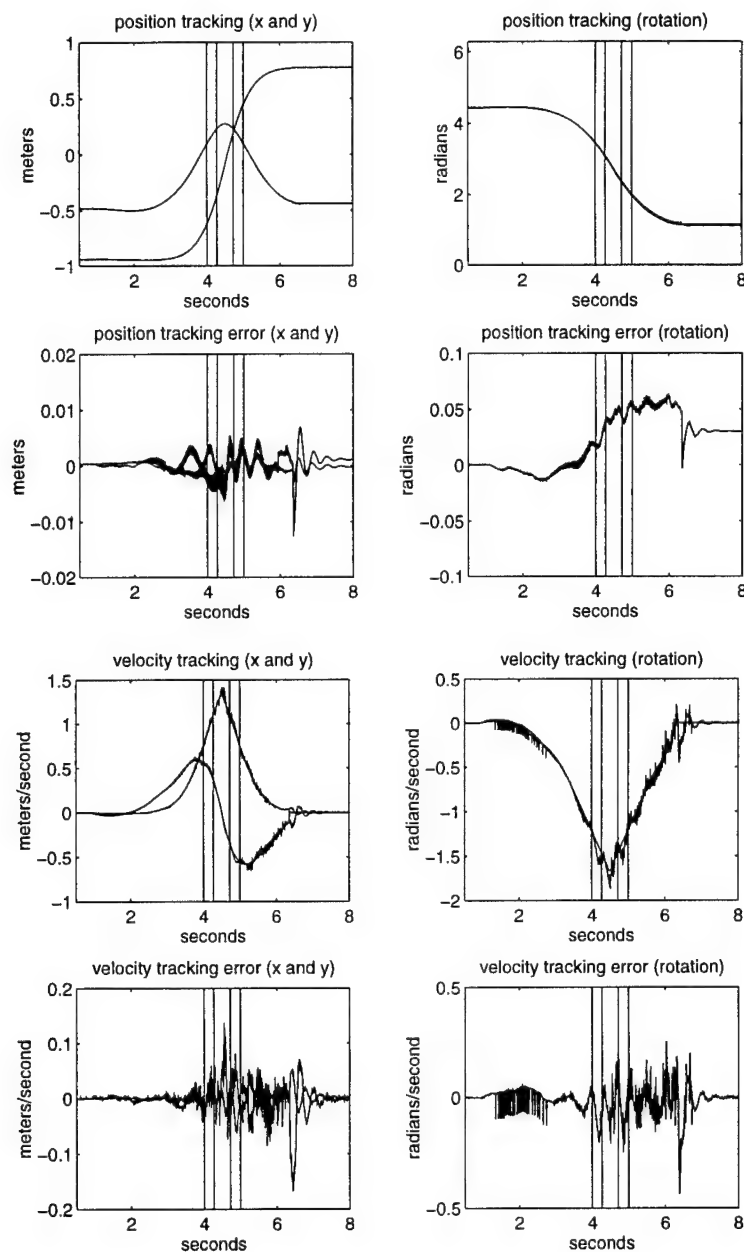
Figure 3.7: **Experimental tracking performance: path through kinematic singularity.**

Tracking performance on X, Y, yaw position and velocities. The above plots illustrate reference and actual trajectories (top line), the tracking error (second line), the reference and actual velocities (third line), and velocity tracking error (fourth line). The vertical lines indicate the region where the arm is at a kinematic singularity. In the center band between the innermost vertical lines, the arm is under pure joint-based control, in the adjacent bands, cartesian and joint control commands are averaged (see Figure 3.2), away from the singularity pure cartesian control is used.
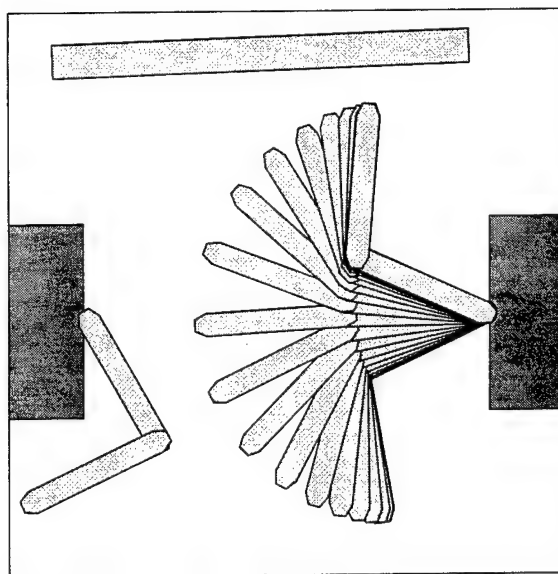
Figure 3.8: **Animation of trajectory through kinematic singularity.**

This figure animates the data collected during the trajectory shown in Figure 3.7.
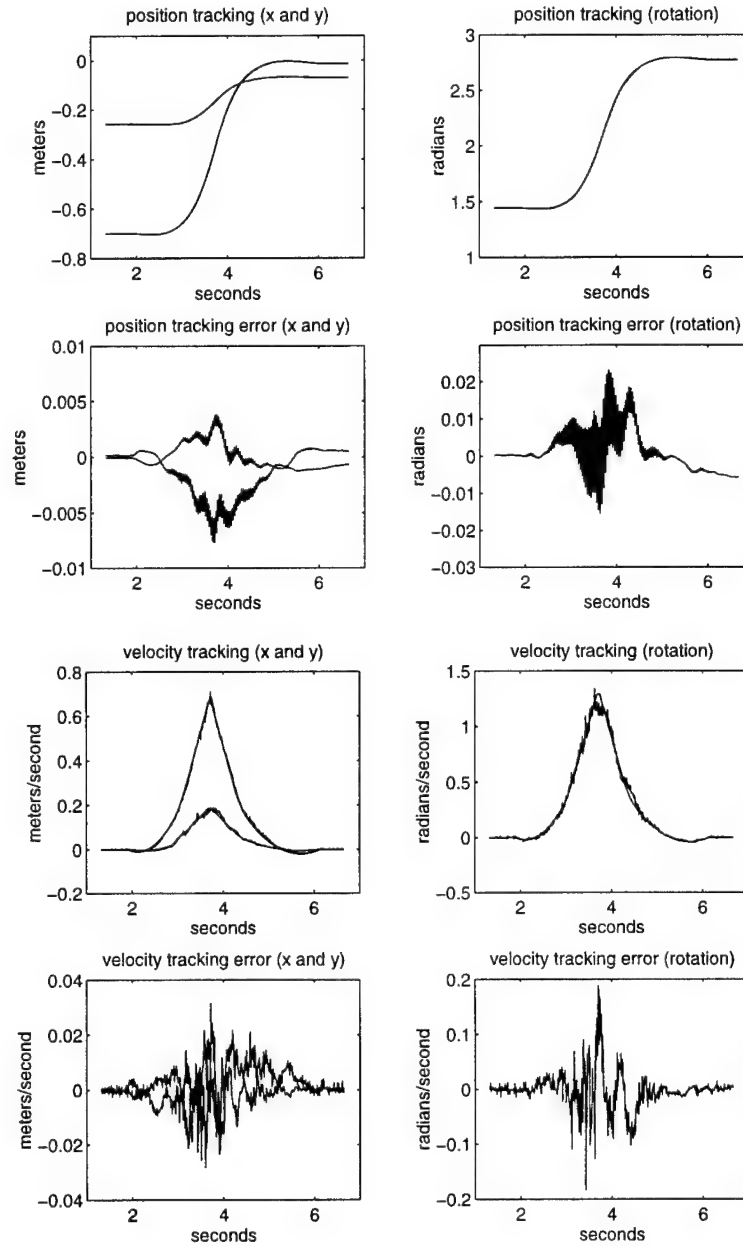
Figure 3.9: **Experimental tracking performance of the object-impedance controller**

Tracking performance on X, Y, yaw position and velocities. The above plots illustrate reference and actual trajectories (top line), the tracking error (second line), the reference and actual velocities (third line), and velocity tracking error (fourth line), for the object trajectory shown in Figure 3.10.
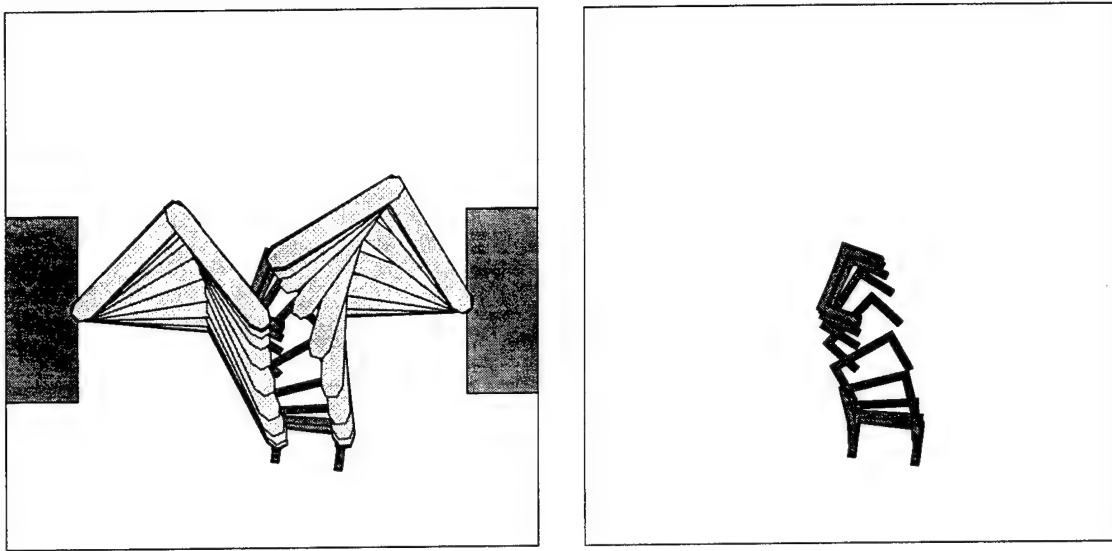
Figure 3.10: **Animation of trajectory followed by the object under cooperative control**

This figure animates the data collected during the trajectory shown in Figure 3.9.

Figure 3.9 illustrates the tracking performance of the OIC for a cooperative object motion in free space animated in Figure 3.10. Comparisons with other object-level controllers and contact experiments have been documented extensively by Schneider and Pfeffer.

Figures 3.11 and 3.12 illustrate the performance of this algorithm to track objects on a conveyor. In all cases the accumulated error is smaller than the accuracy required for a successful capture (about 1 cm).
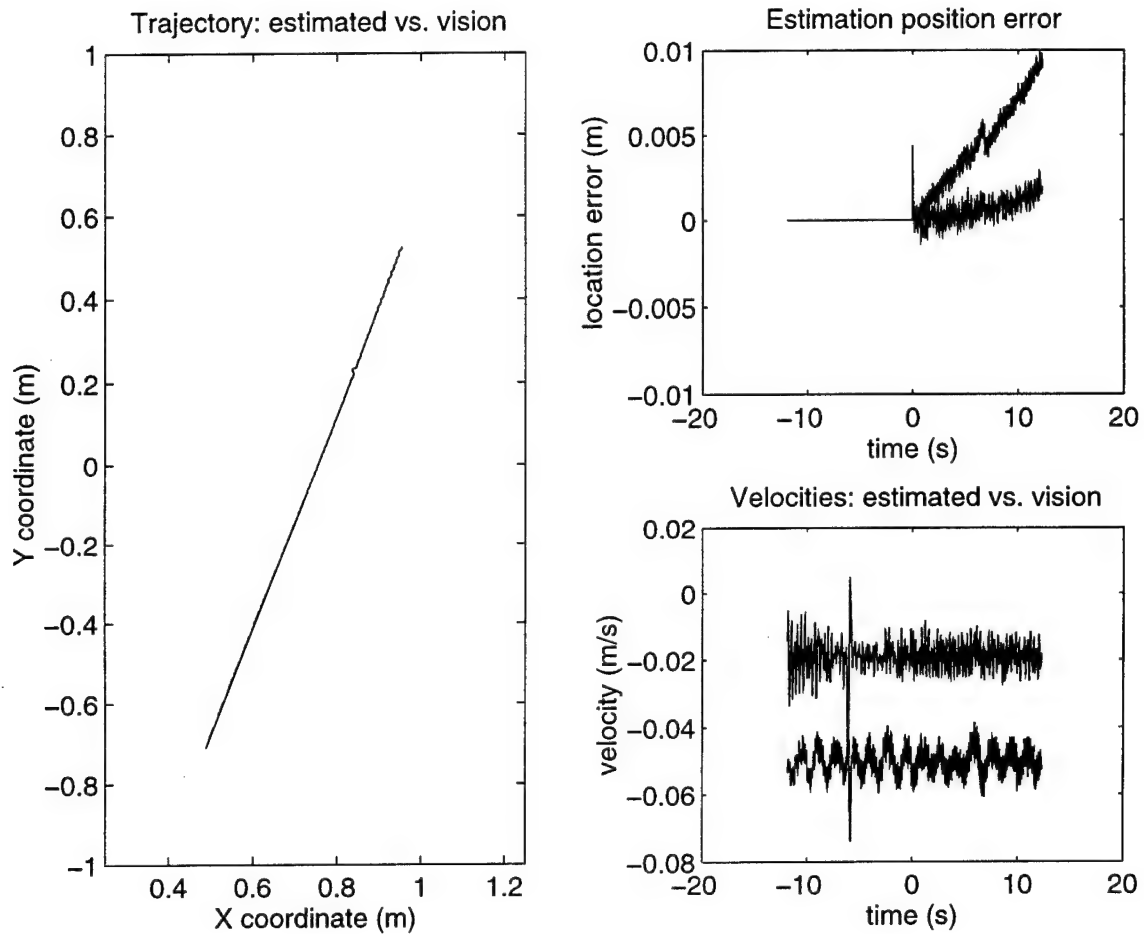
Figure 3.11: **Tracking performance of objects on the conveyor at 5 cm/sec**

The above figures show the performance of the sensor-integration algorithm for the nominal conveyor speed of 5 cm/s. At time t=0, the sensor-integration software was misled into believing that the vision had lost track of the object, so that the difference between the actual vision state and the one derived from the conveyor sensors can be compared. The position discrepancy is on the order of 1 cm even after 0.5 m travel without vision information, this is the limit on the gripper tolerance for a successful capture. The plots on the right illustrate positions and velocities for the X and Y coordinates.
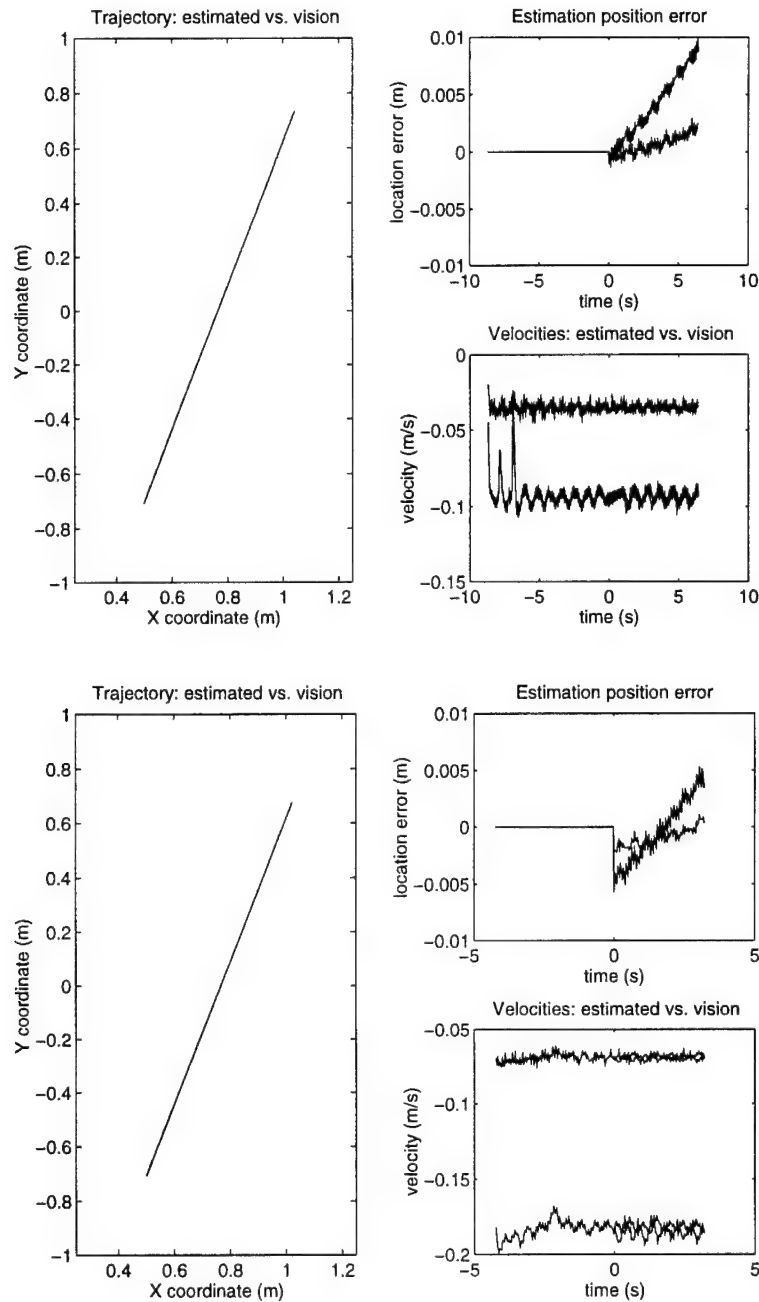
Figure 3.12: **Tracking performance of objects on the conveyor at 10 and 20 cm/s**

The above figures show the performance of the sensor-integration algorithm for faster conveyor speeds: 10 cm/s and 20 cm/s. The approach used is the same explained in Figure 3.11: the sensor-integration algorithm believes the vision has lost track of the object at time t=0. As in 3.11, the the position discrepancy is on the order of 1 cm after 0.5 m displacement. This is mostly due to the error in the orientation of the conveyor as measured by the vision system.

Chapter 4

On-Line Computation Distribution Architectures

Our research addresses technical issues arising when computationally complex algorithms are embedded in a real-time framework. To illustrate these issues we consider two particular problem domains: *object manipulation by autonomous multi-arm robots* and *navigation of multiple autonomous mobile robots in an incompletely known environment.* These two problems raise a number of generic issues directly related to the general theme of our research: motion planning is provably a computationally hard problem and its outcomes, motion plans, are executed in a dynamic world where various sorts of contingencies may happen.

The ultimate goal of our investigation, concerning the two problem domains mentioned above, is to both provide real-time controllers with on-line motion reactive planning capabilities and build experimental robotic systems demonstrating such capabilities. Moreover, in accomplishing this goal, we expect to elaborate general guidelines for embedding a capability requiring provably complex computations into a real-time framework.

This quarterly report covers work done towards this goal during the period of October through December 1993. During this period, our work addressed the following areas:

1. Distribution of Path Planning, by Tsai-Yen Li.

2. New Methods for Fast Path Planning, by Tsai-Yen Li.

3. Parallelization of Path Planning, by Lydia Kavraki.

4. Multi-Arm Manipulation Planning in 3D, by Yotto Koga.

5. Experiments in Manipulation Planning, by Tsai-Yen Li.

6. Landmark-Based Mobile Robot Navigation, by Anthony Lazanas, Craig Becker, Byung-Ju Kang and Ken Tokusei.

7. Mobile Robot Navigation Toolkits and Simulator, by Craig Becker and David Zhu.

Areas 1 through 5 are mainly related to the first problem domain, i.e., *object manipulation by autonomous multi-arm robots.*

Areas 6 through 8 are mainly related to the second problem domain, i.e., *navigation of multiple autonomous mobile robots in an incompletely known environment.*

Participating Ph.D. Students: Craig Becker, Lydia Kavraki, Yotto Koga, Anthony Lazanas, Tsai-Yen Li.

Participating Master Students: Ken Tokusei, Byung-Ju Kang.

Participating Staff: David Zhu.

4.1   Distribution of Path Planning
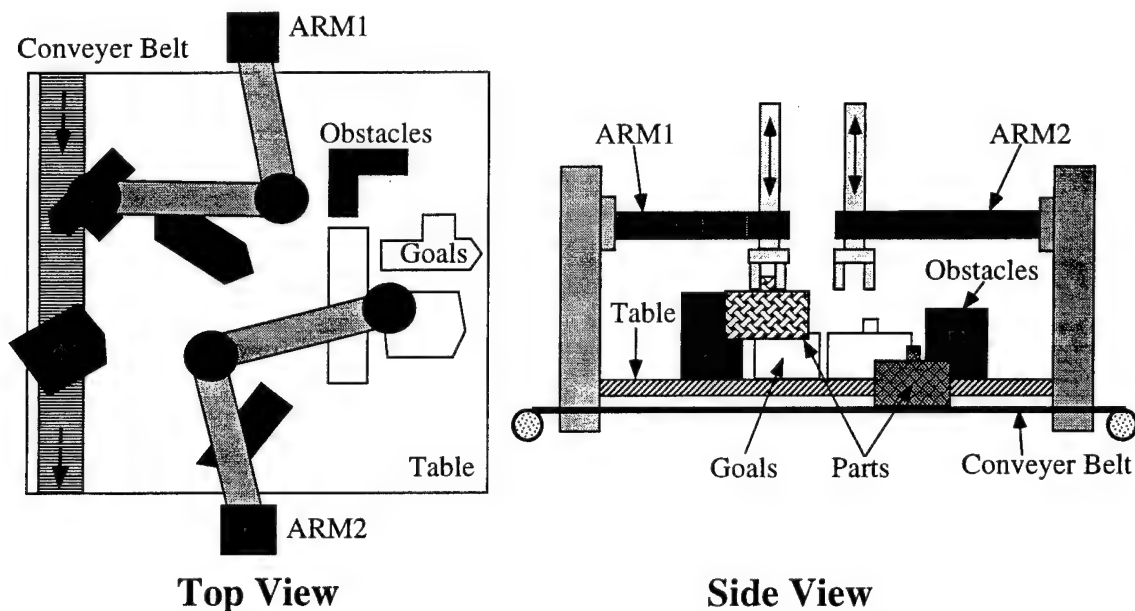

Figure 4.1: **Two-arm robotic cell**

During this quarter, we have continued our research in applying the distribution methodology described in the previous reports to our demonstrative scenario (see Figure 4.1). Recall that there

are two robot arms and multiple moving objects in our scenario. Each arm and object contribute some dimensions (or DOF) to the overall search space where the solutions to our problem reside. Due to the high dimensionality of the composite search space, we cannot treat the problem on-line in a general manner. Instead, we use the distribution principles that we have identified in several different levels of planning to achieve on-line planning performance.

First, at the task level, we distribute the planning over problem approximation by decomposing the multi-arm problem into two subproblems each of which considers one arm only. For now, we assume that every object only requires one arm to grasp. When there is more than one task to plan for, we plan only for one task at a time using the following principle. We first choose one task to plan for according to some prescribed criteria and the found path (if there exists one) is sent for execution immediately. We then plan for another task under the constraint of the arm that is moving along the previously planned path.

For the one-arm manipulation task identified above, we further distribute the planning over time by decomposing it into two subtasks: grasping subtask (catching a moving object), and delivering subtask (delivering the caught object to the goal possibly with regrasping). The main reason for this decomposition is due to the time constraint associated with the motion of grasping a moving object which would leave the work space of the arms in some limited time.

With these two levels of decompositions mentioned above, we identify four distinct subproblems that we will investigate under the distribution:

- catching a moving object when the other arm is free or available.

- catching a moving object when the other arm is already moving.

- delivering an object when the other arm is free.

- delivering an object when the other arm is not free.

In the next section about fast path planning algorithms, we will describe the planning primitives developed for solving these subproblems.

4.2   New Methods for Fast Path Planning

In the previous section we decomposed the overall problem in our scenario into subproblems solvable in a space of dimension 3 or less. During this quarter, we have also developed a planning primitive to solve the first subproblem: Plan a grasp motion for one arm (ARM1) to catch a part on a conveyor when the other arm (ARM2) is available.

One approach to plan ARM1's motion is by treating ARM2 as a static obstacle and searching for a path in ARM1's C-space (a 2D space under our assumption). However, since the reachable regions of the arms overlap, it often happens that ARM2 is in a configuration obstructing ARM1's motion. In this case, the planner may fail to find a path for ARM1 or, even if a path exists, the length of

this path may be too long for ARM1 to catch the moving part in time. As a result, the primitive using this approach is a rather weak one. A better approach is to make use of the fact that ARM2 is available and make ARM2 move out of ARM1's way whenever ARM2 is obstructive to ARM1.

We formalize this problem in the so called Configuration-Time Space (CT-space) where we augment C-space with an additional dimension time. Under our assumptions, the C-space for an arm at any given time instant is a 2D space whose C-obstacles is a function of the other arm's configuration. Therefore, the dimensions of CT-space is three in this case.

This planning primitive is an iterative two-step process: finding the most efficient motion for ARM1 to catch the moving part at a predicted goal, finding the necessary motion for ARM2 to move out ARM1's way, and then deciding whether this coordinated path is feasible for execution. Since the part is moving, the goal configurations for ARM1 correspond to a curve segment in ARM1's CT-space (CT_1). In the beginning of the iterative process, the last goal point in this curve is used to plan for ARM1's motion. In the first step, we plan the most efficient motion (e.g. a straight-line motion in ARM1's joint space satisfying some maximum velocity constraint) for ARM1 to reach ARM2's goal configuration disregarding the existence of ARM2. In the second step, we map the path of ARM1 into some forbidden region (called CT-obstacles) in the ARM2's CT space (CT_2). A path for ARM2 is then planned such that it stays in the free space of its CT space for the duration of the ARM1 path, i.e., it moves away to accommodate the motion of ARM1. If a path is found and its scheduled starting time (t_s) is later than the current time (t_c) the path is sent for execution immediately. However, if the starting time is much greater than the current time, say greater than a threshold, the planner will try to improve the plan by choosing an earlier goal and runs the two-step planning process again to find a path that can be started earlier. Whenever the current time equals the starting time of the current path (if any), this improvement process is abandoned, and the current path is immediately sent for execution.

We have conducted several experiments of this primitive in a simulated environment to verify and evaluate the algorithm. The results show that the primitive is very efficient and the primary cost for the planning is in the second step of each iteration. A typical planning time for an iteration is around 40 ms on a DEC Alpha workstation and it normally takes 2-3 iterations before a satisfactory path is found.

4.3   Parallelization of Path Planning

During this quarter we continued working on the implementation of the path planning approach that we have developed over the past two quarters. This approach consists of a preprocessing and a path planning stage. In the preprocessing stage, which is done only once for a given workspace, a network is constructed in the free part of the configuration space (C-space). The network nodes correspond to randomly selected collision free configurations and the network edges to simple collision free motions between the nodes. In the planning stage, we connect any initial and final configuration of the robot to two nodes in the network and compute a path through the network to connect these

nodes. We would like to be able to do path planning very fast, after the computationally expensive preprocessing step.

So far, we have concentrated on the generation of a network in the C-space that captures the connectivity of the free C-space as well as possible. As described in the previous report, we first generate a number of random nodes and then use a simple and fast path planner to connect neighboring nodes (according to a distance measure in C-space). We compute the connected components of the resulting graph. We have observed that for examples that involve 7 to 10 dof robots moving in constrained environments a few large components are present at the end of this step, even if the free C-space is connected. We would like to obtain one single connected component in the free C-space, if the latter is connected.

In order to connect produced components to a single component in the free C-space we have two choices. We can (1) use a more sophisticated path planner to obtain connections between nodes in different components or (2) increase the number of generated random nodes (N) and hope that the components will get connected with our simple path planning techniques.

First we tried (1). We used the Randomized Path Planner (RPP) to connect configurations in different components that are "close" according to our measure of distance in C-space. In many cases we achieved connections in reasonable time. However, there are examples where RPP takes long to produce a path.

Then we experimented with (2) above. Increasing N results in producing one connected component in easy cases. But for the examples we were interested in, this did not happen unless N became very large. Also, we have noticed repeatedly that there are areas of the workspace where it is difficult to add nodes: the number of random nodes created in these areas increases very slowly as the total number of node increases.

We believe that it will be to our advantage to add nodes non-randomly at this stage and try to enhance the information we have about these areas. One way to do this is to expand all the configurations belonging to small components. By expansion we mean the creation of a random node in the neighborhood of the expanded node. This can be done by varying each dof of the robot in an interval centered around its current value. Intuitively the expansion of the small components leads to an explosion of the area covered by the component. (We consider as small any component with less than 30% of the total nodes.) The new nodes created by expansion are tried for connections with the previous nodes as before, that is we try to connect each of them to its neighbors using our simple path planner. After this, we again compute the connected components of the graph.

Indeed preliminary experiments have shown that the components computed after the enhancement are denser and seem to cover the space better. In many cases we obtained one significant connected component in the graph at the end of the preprocessing step. In other cases, where we still needed to use RPP, we observed that running times seemed to decrease.

During next quarter we plan to work more on the enhancement idea outlined above. It may be possible to find heuristics that better identify the difficult parts of the C-space and other expansion techniques that are more efficient that what we use now. We believe that enhancement is a key

element for the success of our method.

4.4   Multi-Arm Manipulation Planning in 3D

Details of our multi-arm manipulation planner for a 3D workspace are given in the first three quarterly reports of '93. The manipulation planner was originally developed for robotics applications but we have found that it will apply to any system of linkages that has an inverse kinematics solution. We are thus, investigating the full range of applications for our manipulation planner. During this quarter we have begun integrating our manipulation planner with a human arm inverse kinematics algorithm. The particular inverse kinematics solution we are considering is the algorithm by Kondo, which is based on the sensory-motor transformation model from neurophysiology. The goal of this particular line of research is to automatically compute realistic human arm motions to complete such high-level task commands as "move the box to the other side of the table". The input to the system would be a model of the environment, the movable object, and the forward and inverse kinematics of the human arms. The output is the desired motions computed by the manipulation planner. We see such a system as being a useful tool for studying the ergonomy of tasks and products.

4.5   Experiments in Manipulation Planning

During the last quarter, we extended our task planner from handling a single object to handling multiple objects with various grasp requirements for the arms. In this quarter, we have finished the implementation of basic functionalities for handling multiple objects in the task planner and experimented it with the newly developed path planning primitive described in the previous section.

The task planner plays the role of a decision maker between the robot and the path planning modules. The task planner keeps a list of objects reported by the world modeler and a list of goals to achieve specified by the user interface. When there are multiple tasks needed to be planned, the task planner assigns priorities to the arms and objects according to some prescribed rules and sends the task (using which arm to grasp which object) with the highest priority to the path planning module to search for a path. For example, when there is a mix of static and moving objects in the work space, the task planner first plans for the static objects that were placed on the table but haven't reached their goals yet. This is due to the fact that these static objects are already in the work space and could be potential obstacles that prevent other moving objects to reach their goals. For moving objects, the task planner always plans for the earliest object that would leave the work space so that it can accomplish as many tasks as possible. When a moving object is reachable by both arms and both arms are available, the task planner always choose the arm which is closer to the starting position of the conveyor belt first and uses the other arm as a backup when the first trial fails.

We have conducted extensive experiments on the task planner with the path planning primitive

mentioned in Section 3. The results show that the task planner is very robust in handling possible disturbances from the outside world (e.g. the on-line changes of the current location or the goal location of an object). When the changes in the state of the world are detected, the task planner can always come up with a feasible path to achieve the new goals or fail gracefully by notifying the user the causes for the failure.

4.6   Landmark-Based Mobile Robot Navigation

During the previous quarters we have developed and implemented several efficient algorithms for landmark-based mobile robot navigation. Mobile robot navigation is perhaps the most crucial problem in mobile robotics. Despite a lot of research effort over the past two decades, the problem still has no satisfactory solution. Prior theoretical studies and experiments with implemented systems tell us that:

- One cannot build a truly reliable system without both making clear assumptions bounding uncertainty and enforcing these assumptions by appropriately engineering the robot and/or its workspace.

- If assumptions are too mild, the planning subproblem is computationally intractable. If assumptions are too strong, engineering is too costly and/or navigation not flexible enough.

Our research investigates the tradeoff between "computational complexity" and "physical complexity" in reliable mobile robot navigation. Our approach consists of:

1. Defining a formal navigation problem with just enough assumptions to make it possible to construct a sound and complete planner that is also computationally efficient.

2. Designing and implementing such a planner, in order to verify that the planner is actually efficient.

3. Engineering a robot and its workspace to enforce the assumptions in the defined problem, in order to verify that the "cost" of such engineering is reasonable.

4. Implementing a navigation algorithm that makes a real robot execute plans generated by the planner, in order to verify that navigation is actually reliable.

This approach also induces a new role for experimentation in robotics: When robot algorithms are proven correct under formal assumptions, the purpose of experimentation shifts from demonstrating that they behave as intuitively expected on a sample of tasks, to verifying that the amount of engineering induced by the assumptions is acceptable.

Our previous quarterly reports describe work on steps 1 and 2 above. In the Spring'93 quarter we started dealing with steps 3 and 4. Our work centered on experimentations necessary to implement the landmark-based motion planner developed during the previous quarter. We decided to use visual landmarks that are to be located on the ceiling in an indoor environment to designate the landmark regions, where the mobile robot is assumed to have no uncertainty in sensing. A CCD camera module is installed on the top of the robot, pointing upward to detect and identify landmarks.

During the summer quarter '93 we implemented a prototype of the landmark-based planning and navigation of a mobile robot in an indoor environment using visual landmarks located on the ceiling. We also improved the design of the landmark to allow larger number of landmarks (up to 512) and implemented a faster recognition algorithm (approximately 600 milliseconds on an 80386-based robot).

During this quarter, we extended the planner slightly to allow a new kind of landmark, called "generalized landmark," that does not require perfect sensing and control anywhere in the landmark area induced by this landmark. We also conducted additional experiments. We found out that, in some circumstances, being able to control directional uncertainty (a facility embedded in our planner) is key to navigation success. One such case when the environment requires us to place landmarks far apart; by spending more time in image processing under a landmark (using sub-pixel treatment), we were able to reduce directional errors by 3 to 5, allowing the robot to reliably navigate in areas sparsely covered by landmarks.

4.7   Mobile Robot Navigation Toolkits and Simulator

Over the past quarters we have developed mobile robot navigation toolkits embedding a variety of functions, such as potential field computation, navigation in potential fields, path planning, landmark-based navigation, static and dynamic localization using environment sensing, etc. We also have developed a simulator for one or several mobile robots operating in the same environment.

Both these toolkits and simulator have been transferred to Nomadic Technology. This company now markets this software (it has sold over 50 mobile robots to a variety of universities, organizations, and companies in the US and abroad). We consider that our research on this topic has attained its objectives. We will not conduct additional research on these topics within this project.

4.8   Summary of Main Results Obtained So Far

1. Identification of several axes for distributing path planning software in an on-line architecture.

2. A documented Randomized Path Planner package has been made available to other research institution on the computer network. Several organizations are using it.

3. Implementation of parallel versions of RPP on a Silicon Graphics 4D/240 multiprocessor machine and on a local-area network of UNIX-based workstations.

4. Definition of a new, FFT-based method to compute obstacles in configuration space.

5. Definition and implementation of a new path planning method (the vector-based planner) to generate paths for robots with many degrees of freedom.

6. Integration of several path planners (RPP, vector-based planner with/without potential fields) in a package distributed over a network of UNIX-based workstations.

7. Partial design and implementation of a new, two-phase path-planning approach, which we call the "randomized roadmap planner."

8. Design and implementation of an optimal-time motion planner for closed-loop kinematic chains.

9. Design and implementation of a randomized three-arm manipulation planner for manipulating an elongated object in a 3D cluttered environment.

10. Design and implementation of a new landmark-based mobile robot planning method. Extension of this planner to deal with controllable uncertainty.

11. Definition of the layout of a software toolkit to efficiently develop new navigation systems. Implementation of several toolkits.

12. Development of a powerful multi-mobile-robot simulator to facilitate the development and debugging of programs for multiple interacting mobile robots.

PhD Defenses:

Anthony Lazanas (Landmark-based navigation planning) successfully passed his PhD orals.

Others:

- J.C. Latombe was elected AAAI Fellow for his contributions to the "Theory and Practice of Robot Motion Planning."

- C.Becker was a member (with 3 other students) of the Stanford team that won the first event at the AAAI-93 Mobile Robot competition, using our NOMAD 200 robot.

4.9 Status

Our research progresses according to schedule.

Chapter 5

Applications and Technology Transfer

It is not possible to develop generic technology without multiple, specific applications to test and refine the ideas and implementations. As such, we are actively seeking sites, both internally and externally to provide the compelling test beds that will make this project succeed. These driving applications span a variety of the most important target users: high-performance control, intelligent machine systems, underwater vehicle command and control, and remote teleoperation. Several of these projects will reach for new limits in advanced technology and system integration; others will address real-world problems in operational systems.

With the reduced funding levels, we will not have the resources to support all of the originally proposed technology evaluation sites. However, we believe these sites are crucial to the development of ControlShell into a viable technology for "real-world" use. Thus, we have actively pursued alternative means of supporting external sites. We have been successful in securing several new test applications. These sites will either function with minimal support, or fund their own support.

This chapter highlights some of the activities of these projects.

The currently-active ControlShell applications are:

- Precision Machining, by The Stanford Quiet Hydraulics Laboratory.

- Underwater Vehicle Control, a joint project between the ARL and the Monterey Bay Aquarium Research Institute.

- Intelligent Machine Architectures, by Lockheed Missiles and Space Corporation.

- Remote Teleoperation, by Space Systems Loral Corporation.

- Space-based Mobile Robot Systems, by several ARL students (NASA-sponsored).

- High-Performance Control of Flexible Structures, by several ARL students (AFOSR-sponsored).

- Space-structure assembly, by NASA Langley Research Center.

- Mobile-robot control by NASA Ames Research Center.

5.1 NASA sites

5.2 MBARI Underwater Vehicle

5.3 Transfer of Planning Technology

- Part of our mobile robot software (simple planner, simulation, landmark vision software) has been ported by Nomadic Technologies, and is part of the software distributed by this company with their mobile robot NOMAD 200.

- J.C. Latombe and L. Kavraki assisted Nova Management, Inc., in building an automated route planner for tanks in support of US Government Contract No. DAAE07-C-93-0026. A prototype version of this planner was successfully demonstrated to Army representatives.

- B.Romney (a PhD student) spent the summer at GM Research Labs in Warren, MI, and implemented a version of assembly planner there. He connected this planner to the UNI-GRAPHICS CAD system.

- R.H. Wilson (a Ph.D. student, then a Research Associate) was hired as a Research Scientist by SANDIA Labs, Albuquerque, NM.

- A. Lazanas, a Ph.D. student involved in this project, has been hired by Salomon Brothers, NY, to apply his expertise in geometric computing to finding good investment strategies.

Publications So Far:

R.I. Brafman, J.C. Latombe, and Y. Shoham, "Towards Knowledge-Level Analysis of Motion Planning," *Proc. of the 11th Nat. Conf. on Artificial Intelligence*, AAAI-93, Washington D.C., July 1993, pp. 670-675.

R.I. Brafman, J.C. Latombe, Y. Moses, and Y. Shoham, "Knowledge as a Tool in Motion Planning Under Uncertainty," *Proc. of the 5th Conf. on Theoretical Aspects of Reasoning About Knowledge*, Monterey, CA, Morgan-Kaufmann Publishers, 1994, pp. 208-224. **Publications So Far:**

L. Kavraki, *Computation of Configuration-Space Obstacles Using the Fast Fourier Transform*, Technical Report, STAN-CS-92-1425, 1992.

L. Kavraki, *Computation of Configuration-Space Obstacles Using the Fast Fourier Transform*, *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Atlanta, GA, 1993.

L. Kavraki, J.C. Latombe, and R.H. Wilson, "On the Complexity of Assembly Partitioning," *Proc. of 5th Canadian Conf. on Computational Geometry*, August 1993, pp. 12-17.

L. Kavraki, J.C. Latombe, and R.H. Wilson, "On the Complexity of Assembly Partitioning," accepted for publication in *Information Processing Letters*.

L. Kavraki and J.C. Latombe, *Randomized Preprocessing of Configuration Space for Fast Path Planning*, Rep. No. STAN-CS-93-1490, Dept. of Computer Science, Stanford University, September 1993.

Y. Koga and J.C. Latombe, "Experiments in Dual-Arm Manipulation Planning," *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Nice, May 1992, pp. 2238–2245.

Y. Koga, T. Lastennet, J.C. Latombe, and T.Y. Li, "Multi-Arm Manipulation Planning," *Proc. of the 9th Int. Symp. on Automation and Robotics in Construction*, Tokyo, June 1992.

J.C. Latombe, "Geometry and Search in Motion Planning," *Annals of Mathematics and Artificial Intelligence*, 8(2-4), 1993.

J.C. Latombe, "Robot Algorithms," *Proc. of the LAAS/CNRS 25th Anniversary Conf.*, Cepadues, Toulouse, France, May 1993, pp. 81–94 (invited conference).

J.C. Latombe, "Robot Algorithms," presented at the WAFR Workshop held in San Francisco,

CA, February 1994. To appear in *Foundations of Robot Algorithms*, D. Halperin, K. Goldberg, J.C. Latombe, and R.H. Wilson (eds.), AK Peters, Wellesley, MA, 1995.

A. Lazanas and J.C. Latombe, *Landmark-Based Robot Navigation*, Rep. No. STAN-CS-92-1428, Dept. of Computer Science, Stanford U., May 1992. Accepted for publication in *Algorithmica*.

A. Lazanas and J.C. Latombe, "Landmark-Based Robot Navigation," *Proc. of the 10th Nat. Conf. on Artificial Intelligence, AAAI-92*, San Jose, July 1992, pp. 816–822.

A. Lazanas and J.C. Latombe, "Landmark-Based Robot Motion Planning," *Proc. of the AAAI Fall Symp.*, Boston, MA. October 1992, pp. 98-103.

A. Lazanas and J.C. Latombe, "Landmark-Based Robot Motion Planning," *Geometric Reasoning for Perception and Action*, C. Laugier (Ed.), Lecture Notes in Computer Science, 708, Springer-Verlag, 1993.

Gerardo Pardo-Castellote and Robert H. Cannon Jr. "Proximate time-optimal parameterization of robot paths," STAN-ARL-92- 88, Stanford University Aerospace Robotics Laboratory, April 1993.

Gerardo Pardo-Castellote, Tsai-Yen Li, Yoshihito Koga, Robert H. Cannon Jr., Jean-Claude Latombe, and Stan Schneider," "Experimental integration of planning in a distributed control system. In *Preprints of the Third International Symposium on Experimantal Robotics*, Kyoto Japan, October 1993.

Gerardo Pardo-Castellote and Stanley A. Schneider. "The network data delivery service: real-time data connectivity for distributed control applications," (to appear in) In *Proceedings of the International Conference on Robotics and Automation*, San Diego, CA, May 1994. IEEE, IEEE Computer Society.

Gerardo Pardo-Castellote and Stanley A. Schneider. "The network data delivery service: A real-time data connectivity system," (to appear) In *Proceedings of the AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service and Space*, Houston, TX, March 1994. AIAA, AIAA.

S. Schneider and R. H. Cannon. "Object impedance control for cooperative manipulation: Theory and experimental results," *IEEE Journal of Robotics and Automation*, 8(3), June 1992. Paper number B90145.

S. A. Schneider and R. H. Cannon. "Experimental object-level strategic control with cooperating manipulators," *The International Journal of Robotics Research*, 12(4):338–350, August 1993.

Howard H. Wang, Richard L. Marks, Stephen M. Rock, and Michael J. Lee. "Task-based control architecture for an untethered, unmanned submersible," In *Proceedings of the 8th Annual Symposium of Unmanned Untethered Submersible Technology*, pages 137–147. Marine Systems Engineering Laboratory, Northeastern University, September 1993.